

Stochastic Local Search for Pattern Set Mining

Muktadir Hossain, Tajkia Tasnim, Swakkhar Shatabda and Dewan M Farid
Department of Computer Science and Engineering, United International University
House # 80, Road # 8A, Dhanmondi, Dhaka-1209, Bangladesh

Email: muktadir00@gmail.com, tajkiamim@gmail.com, swakkhar@cse.uiu.ac.bd, dewanfarid@cse.uiu.ac.bd

December 19, 2014

Abstract

Local search methods can quickly find good quality solutions in cases where systematic search methods might take a large amount of time. Moreover, in the context of pattern set mining, exhaustive search methods are not applicable due to the large search space they have to explore. In this paper, we propose the application of stochastic local search to solve the pattern set mining. Specifically, to the task of concept learning. We applied a number of local search algorithms on a standard benchmark instances for pattern set mining and the results show the potentials for further exploration.

1 Introduction

There have been growing interest in the field of *pattern set mining* in stead of pattern mining in the recent years [1]. One of the most important task in pattern set mining is to find a particular set of patterns in data that successfully partitions the dataset and discriminates the classes from one another [2]. This is called *concept learning* task in the literature [2]. Such pattern sets are desirable incase of selecting a particular small set of patterns from a large dataset, where traditional pattern mining algorithms fail to produce good results.

In the concept learning task, we are given a set of classes and a set of patterns or features. The task is to select a small set of the features or the patterns so that the classification accuracy is maximized. This is a complex combinatorial optimization task. Most of the algorithms that are applied to solve the problem earlier in the literature are mostly exhaustive or greedy in nature [2]. Declarative frameworks like constraint programming methods [2, 3] have gained some significant success. However, with increasing problem size, these methods struggle to produce good quality solutions within a short period of time. On the other hand, local search methods in general can quickly find good quality solutions and have been very effective to find satisfactory results for many combinatorial optimization problems.

In this paper, we propose to apply a large variety to local search algorithms to solve the concept learning task in the context of pattern set mining. The set of algorithms that we use includes random walk, random valid walk, hill climbing, hill climbing with restart and genetic algorithm. We performed our experiments to solve standard benchmark datasets extensively used by the researchers in the literature. The key contributions in the paper are as follows:

- Demonstrate the overall strength of stochastic local search methods solving the pattern set mining task.

Table 1: A small example dataset containing four items and five transactions.

Transaction		A	B	C	D	Class
Id	ItemSet					
t_1	$\{A,B,D\}$	1	1	0	1	+
t_2	$\{B,C\}$	0	1	1	0	+
t_3	$\{A,D\}$	1	0	0	1	+
t_4	$\{A,C,D\}$	1	0	1	1	-
t_5	$\{B,C,D\}$	0	1	1	1	-

- Perform a comparative analysis of various local search algorithm and analysis of their relative strength compared with each other.

The rest of the paper is organized as follows: Section 2 describes the concept learning task in the context of pattern sets mining; Section 3 reviews the related work; Section 4 describes the algorithms used; Section 5 discusses and analyzes the experimental results; and finally, Section 6 presents our conclusions and outlines our future work.

2 Preliminaries

In this section, we briefly describe the problem model that we use. We adopt the itemset mining [3] setting used earlier by Guns et al. in [2]. In the concept learning task we are given a database of transactions, \mathcal{D} . Which is a binary matrix, meaning all items are either 0 or 1. The columns of this binary matrix corresponds to a set of patterns or items, \mathcal{I} and the rows corresponds to actual data items or transactions. The set of transactions is denoted by \mathcal{T} . For example consider the dataset given in Table 1. Here, we have four items in the itemset, $\mathcal{I} = \{A, B, C, D\}$ and five transactions in the transaction set, $\mathcal{T} = \{t_1, t_2, t_3, t_4, t_5\}$. The itemsets or pattern sets and the transaction sets are generally represented by binary vectors. The *coverage* $\varphi_{\mathcal{D}}(I)$ of an itemset I consists of all transactions in which the itemset occurs:

$$\varphi_{\mathcal{D}}(I) = \{t \in \mathcal{T} | \forall i \in I : \mathcal{D}_{ti} = 1\}$$

For example, given an itemset, $I = A, D$, it is represented as $\langle 1, 0, 0, 1 \rangle$ and the the coverage is $\varphi_{\mathcal{D}}(I) = \{t_1, t_3, t_4\}$ which is represented by $\langle 1, 0, 1, 1, 0 \rangle$. Support of the itemset is $Support_{\mathcal{D}}(I) = 3$. Where, Support of an itemset is the size of its coverage set, $Support_{\mathcal{D}}(I) = |\varphi_{\mathcal{D}}(I)|$.

Any item set I is *closed* over T , if no other supersets of this itemset covers the same transaction set T as I . Formally, closedness $\psi(I, T)$ is defined as the following constraint:

$$\psi(I, \mathcal{T}) = 1 \Leftrightarrow (\forall i \in \mathcal{I} : I_i = 1 \Leftrightarrow \bigwedge_{t \in \mathcal{T}} (\mathcal{D}_{ti} = 1 \vee T_t = 0))$$

For example, the itemset $I_1 = A, D$ is closed and the itemset, $I = A, C$ is not closed. Each of the transactions in the dataset are associated with two distinct classes: positive transactions \mathcal{T}^+ and negative transactions \mathcal{T}^- . Now, we can

define the *accuracy* of an itemset or patternset which is defined by its coverage set T as following:

$$accuracy(T) = \sum_{t \in \mathcal{T}^+} T_t - \sum_{t \in \mathcal{T}^-} T_t$$

In pattern set mining, we are interested to find k -pattern sets [4]. A k -pattern set Π is a set of k tuples, each of type $\langle I^p, T^p \rangle$. The pattern set is formally defined as following:

$$\Pi = \{\pi_1, \dots, \pi_k\}, \text{ where, } \forall p = 1, \dots, k : \pi_p = \langle I^p, T^p \rangle$$

The concept learning task is to find a set of itemsets or patterns that together cover as many positive transactions as possible, while covering only few negative transactions. This can be formalized as the following optimization problem:

$$\underset{\Pi, T}{\text{maximize}} \text{ accuracy}(T)$$

where,

$$\forall t \in \mathcal{T} : T_t = (\bigvee_{p=1 \dots k} T_t^p)$$

$$\forall \pi \in \Pi : \varphi_{\mathcal{D}}(I^p) = T^p$$

$$\forall \pi \in \Pi : \psi(I, \mathcal{T}^+) = 1$$

In this paper, we attempt to solve this problem using different types of stochastic local search algorithms: random walk, random valid walk, hill climbing, hill climbing with restart and genetic algorithm.

3 Related Works

The frequent itemset mining task was first proposed in [5] followed by a large number of algorithms and their application is various fields. In the the pattern mining problem [6], we are interested to find patterns in the dataset that are correlated [7], discriminative [8], contrast [9], diverse [10] etc. A large variety of algorithms has been proposed to solve the pattern set mining in the last decade [1].

Recent advancements in the field shows the use of declarative programming frameworks i.e. constraint programming [3]. State-of-the-art constraint solvers i.e. COMET [11] are used to solve the pattern set mining and related tasks [2]. It opens a huge research area by combining data mining or machine learning tasks with constraint programming [12, 13]. However, most of these methods use systematic search methods to search and shrink the solution space. Most of the systematic search algorithms are exhaustive in nature and require huge amount of time and resources to solve the problem efficiently. Where as greedy methods does not guarantee optimality. The necessity of efficient global optimization algorithms and effective search heuristics for such problems is higher than ever.

In [10], stochastic search algorithms are applied to solve a related problem to find diverse pattern sets. However, Guns et al. [2] applied large neighborhood search, using constraint programming platform to solve both of the concept learning task and the diverse pattern set mining problem.

4 Our Approach

In this section, we describe the algorithms that we have implemented to solve the concept learning task. The basic framework contains two important functions that calculates the coverage and checks the closedness or maximal property of the itemsets or patternsets.

Algorithm 1: `getCoverage(ItemSet I , TransactionSet \mathcal{T})`

```

1  $coverage = [0, \dots, 0]$ 
2 for each transaction  $t \in \mathcal{T}$  do
3   if all items in  $I$  appears in  $t$  then
4      $coverage[t] = 1$ 
5 return  $coverage$ 
```

The *getCoverage* algorithm is given in Algorithm 1. It simply searches for occurrences of items in an itemset in all transactions in the dataset and returns them all. The next important function is the *isClosed* function that verifies the maximal property of an itemset. The algorithm is pictured in Algorithm 2.

Algorithm 2: `isClosed(ItemSet I , Database \mathcal{D})`

```

1  $s_0 = Support_{\mathcal{D}}(I)$ 
2 for each item  $i \in \mathcal{D}$  do
3   if  $i \notin I$  then
4      $I_n = I \cup \{i\}$ 
5      $s_n = Support_{\mathcal{D}}(I_n)$ 
6     if  $s_n \geq s_0$  then
7       return false
8 return true
```

The calculation of accuracy is done by calculating the number of positive transactions, covered by any of the coverage sets of the itemsets in the pattern set and subtracting the number of negative transactions covered by any of the coverage sets of the itemsets, in the pattern set. The algorithm for calculating accuracy in Algorithm 3.

Rest of the section describes the algorithms that we have implemented on this setup using these functions.

4.1 Random Walk

The random walk algorithm is similar to Monte Carlo simulations in nature. The pseudo-code for the algorithm is given in Algorithm 4. This algorithm continuously updates the maximum accuracy and the best solution Π^* and then returns the best solution found before the algorithm times out.

4.2 Random Valid

The next algorithm we call random valid. It starts by initially generating a valid patterns set. A pattern set is valid if the itemsets are all maximal. The termination criteria is similar to the previous algorithm random walk. In this version, the algorithm make some random changes in the current solution and accepts the new solution only if the resulting pattern set is valid in each iteration. Our random valid algorithm is given in Algorithm 5.

Algorithm 3: getAccuracy(PatternSet Π , Database \mathcal{D})

```
1  $\mathcal{C}^+$ : set of positive coverages
2  $\mathcal{C}^-$ : set of negative coverages
3 for each itemset  $I \in \Pi$  do
4    $\mathcal{C}^+[i] = \text{getCoverage}(I, \mathcal{T}^+)$ 
5    $\mathcal{C}^-[i] = \text{getCoverage}(I, \mathcal{T}^-)$ 
6  $posN = 0$ 
7 for each transaction  $t \in \mathcal{T}^+$  do
8   for each coverage  $c \in \mathcal{C}^+$  do
9     if  $c[t] == 1$  then
10        $posN++$ 
11       break
12  $negN = 0$ 
13 for each transaction  $t \in \mathcal{T}^-$  do
14   for each coverage  $c \in \mathcal{C}^-$  do
15     if  $c[t] == 1$  then
16        $negN++$ 
17       break
18 return  $posN - negN$ 
```

Algorithm 4: RandomWalk()

```
1  $maxAccuracy = -\infty$ 
2  $\Pi^* = \phi$ 
3 while timeout do
4    $\Pi = \text{randomly create a k-itemset}$ 
5    $isMaximal = true$ 
6   for each itemset  $I \in \Pi$  do
7     if  $isClosed(I, \mathcal{T}^+) == false$  then
8        $isMaximal = false$ 
9       break
10  if  $isMaximal == true$  then
11     $accuracy = \text{getAccuracy}(\Pi, \mathcal{D})$ 
12    if  $accuracy > maxAccuracy$  then
13       $maxAccuracy = accuracy$ 
14       $\Pi^* = \Pi$ 
15 return  $\Pi^*$ 
```

Algorithm 5: RandomValid()

```
1  $\Pi = \text{generate a valid pattern set with } k \text{ items}$ 
2  $\Pi^* = \Pi$ 
3  $maxAccuracy = \text{getAccuracy}(\Pi, \mathcal{D})$ 
4 while timeout do
5    $\Pi_t = \text{make random change in } \Pi$ 
6   if  $isValid(\Pi_t, \mathcal{D})$  then
7      $accuracy = \text{getAccuracy}(\Pi_t, \mathcal{D})$ 
8     if  $accuracy > maxAccuracy$  then
9        $maxAccuracy = accuracy$ 
10     $\Pi^* = \Pi_t$ 
11     $\Pi = \Pi_t$ 
12 return  $\Pi^*$ 
```

4.3 Hill Climbing

The hill climbing algorithm is similar to the random valid algorithm. But it differs in how the next candidate solution is accepted at Lines 10-11. The new candidate solution is accepted only if it does not decrease the accuracy of the current candidate pattern set. The algorithm is given in Algorithm 6.

Algorithm 6: HillClimbing()

```

1  $\Pi$  = generate a valid pattern set with  $k$  items
2  $maxAccuracy = getAccuracy(\Pi, \mathcal{D})$ 
3 while timeout do
4    $\Pi_t$  = make random change in  $\Pi$ 
5   if isValid( $\Pi_t, \mathcal{D}$ ) then
6      $accuracy = getAccuracy(\Pi, \mathcal{D})$ 
7     if  $accuracy \geq maxAccuracy$  then
8        $maxAccuracy = accuracy$ 
9        $\Pi = \Pi_t$ 
10 return  $\Pi^*$ 

```

4.4 Hill Climbing with Restart

The hill climbing algorithm pictured in Algorithm 6 is greedy in nature as it only accepts non-decreasing steps, in terms of accuracy. However, such strategy quickly leads the algorithm into local minima and cannot improve further. To tackle this situation, we propose the next algorithm which we call hill climbing with restart. It denoted by *HC + Restart* in the rest of the paper. The pseudo-code for *HC + Restart* is given in Algorithm 7. This algorithm is similar to that of hill climbing but it keeps track of the non improving steps and then it restarts the search after the number of non improving steps, crosses the limit of a parameter *threshold*. The value of *threshold* was kept 100 for these experiments.

Algorithm 7: HillClimbingWithRestart()

```

1  $\Pi$  = generate a valid pattern set with  $k$  items
2  $\Pi^* = \Pi$ 
3  $maxAccuracy = getAccuracy(\Pi, \mathcal{D})$ 
4  $nonImprovingSteps = 0$ 
5 while timeout do
6    $\Pi_t$  = make random change in  $\Pi$ 
7   if isValid( $\Pi_t, \mathcal{D}$ ) then
8      $accuracy = getAccuracy(\Pi, \mathcal{D})$ 
9     if  $accuracy > maxAccuracy$  then
10       $maxAccuracy = accuracy$ 
11       $\Pi^* = \Pi_t$ 
12       $nonImprovingSteps = 0$ 
13   else
14      $nonImprovingSteps++$ 
15     if  $nonImprovingSteps \geq threshold$  then
16        $\Pi$  = generate a valid pattern set with  $k$  items
17        $nonImprovingSteps = 0$ 
18   else
19      $\Pi = \Pi_t$ 
20 return  $\Pi^*$ 

```

4.5 Genetic Algorithm

The single point search methods often starts from a solution in the search space and do not have much chance of divergence. Which is needed to cover the huge search space for global optimization problem. For this reason, multi-point search algorithms are preferred. They keep a number of solutions in the population and runs single point search in parallel. Moreover, combinations of the individuals in the population helps the search to create high quality solutions that resembles the natural process of evolution of species. One such algorithm is genetic algorithms. The genetic algorithm that we propose here keeps a population of pattern sets and then iteratively recombines and mutates the individuals to create new solutions. The pseudo-code for our genetic algorithm is given in Algorithm 8.

Algorithm 8: Genetic Algorithm()

```

1  $p$  : population size
2  $\mathcal{P}$  = generate  $p$  valid pattern sets
3 while timeout do
4    $childCount = 0$ 
5    $\mathcal{P}_c = \{\}$ 
6   while  $childCount < p$  do
7      $\Pi_1 = \text{selectParentAtRandom}(\mathcal{P})$ 
8      $\Pi_2 = \text{selectParentAtRandom}(\mathcal{P})$ 
9      $\Pi_c = \text{crossover}(\Pi_1, \Pi_2)$ 
10    if  $\text{isValid}(\Pi_c)$  then
11       $\mathcal{P}_c = \mathcal{P}_c \cup \{\Pi_c\}$ 
12       $childCount++$ 
13   $\mathcal{P}_m = \{\}$ 
14  for each  $\Pi \in \mathcal{P}_c$  do
15    while true do
16       $\Pi_m = \text{make random change in } \Pi$ 
17      if  $\text{isValid}(\Pi_m)$  then
18         $\mathcal{P}_m = \mathcal{P}_m \cup \{\Pi_m\}$ 
19      break
20   $\mathcal{P} = \text{selectBest}(\mathcal{P} \cup \mathcal{P}_c \cup \mathcal{P}_m)$ 
21 return globalBest

```

In our genetic algorithm, the population is initialized by a number of valid pattern sets. In each generation of the genetic algorithm, the algorithm goes through two phases: crossover and mutation. In the crossover phase, two parent individuals are chosen randomly from the population and a child individual is created. This new individual is created using the one point crossover operation. After the validity of the individual checking, only a valid solution is added to the crossover population. This process iterates until the crossover population size is equal to the population size. Then the mutation phase begins. From each of the valid individuals in the crossover population a valid individual is created and added to the mutation population list. At the end of the mutation phase best solutions from these three population lists are selected for the next generation.

5 Experimental Results

We have implemented the algorithms in Ruby language and have run our experiments on an Intel core i3 2.40 GHz machine with 6 GB ram running 32bit Ubuntu 14.04 operating system.

5.1 Dataset

The benchmark datasets that we use in this paper are taken from UCI Machine Learning repository [14] and originally used in [2]. The datasets are available to download freely from the website: <https://dtai.cs.kuleuven.be/CP4IM/datasets/>. The benchmarks are given in Table 2 with their properties.

Table 2: Description of benchmark datasets.

Data set	Items	Transactions	Density	Class Distribution
zoo-1	36	101	44%	41%
hepatitis	44	137	50%	81%
lymph	60	148	38%	55%
heart-cleveland	95	296	47%	54%
vote	48	435	33%	61%
primary-tumor	31	336	48%	24%

5.2 Results

We performed experiments by running different algorithms on each of the benchmark dataset and reported accuracy in Table 3. Due to the unavailability of COMET, we were unable to compare the search performance with the large neighborhood search used in [2]. Each algorithm were given 10 minutes to finish and the best accuracy found during the runtime is reported in the table. The pattern size setting was varied to see the effect of the size parameter on the performance of the algorithms. The value of the parameter k was varied for the values $\{2, 3, 4\}$. The best values in each row are shown in bold faced fonts.

5.3 Analysis

From the results reported in Table 3, it is evident that genetic algorithms work better than other algorithms for most of the datasets for different pattern sizes. However, in a few cases hill climbing with restart (denoted by HC+Restart in the table) works better. However, the margin is not much higher. The random walk Monte Carlo simulation performs poorly since there not much of intelligent decisions taken during the search. The performance of the random valid search improves since random walk wastes a good amount of time finding valid pattern sets that are closed over the positive transaction set. The hill climbing search quickly gets stuck into the local maximum and once stagnates, it can not improve further due to lack of escaping mechanism. The situation improves when we add random restart strategy along with the hill climbing search. The performance of the algorithms somehow varies with the size of the pattern sets. The situation is pictured in Figure ?? . In this figure, accuracy of the search algorithms are shown as vertical bars for all the datasets for different pattern set sizes. It is evident from the bar diagrams that the increase in the pattern set size favors the genetic algorithm.

We also depict the performance of the search algorithms in Figure 7. It shows the progress of different search algorithms in 10 minutes for a single run of the *hepatitis* dataset. The random walk algorithm lacking intelligent decision through the search can not improve much. Where as, hill climbing improves very quickly by taking greedy best choice moves. However, if it gets stuck and the improvement is possible with random restarts. Genetic algorithm shows continuous improvements

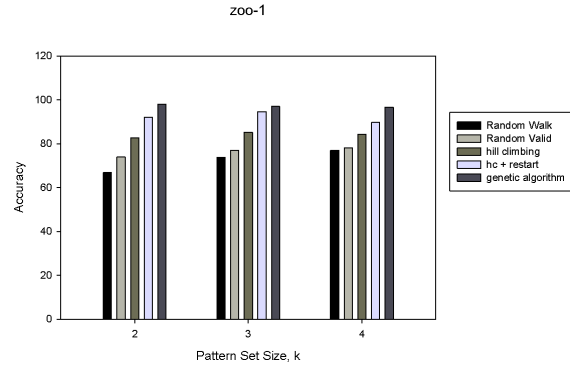


Figure 1: Bar diagram showing comparison of accuracy achieved by different algorithms for various sizes of pattern sets, $k = 2, 3, 4$.

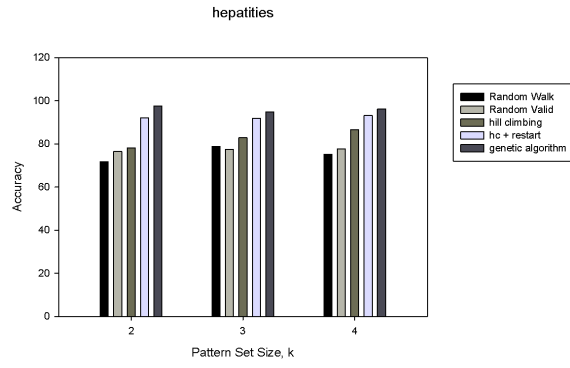


Figure 2: Put a title here

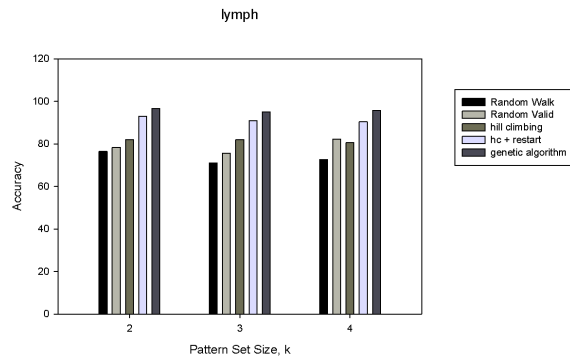


Figure 3: Put a title here

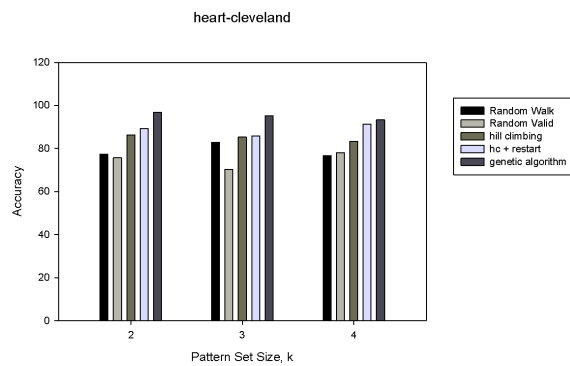


Figure 4: Put a title here

Table 3: Accuracy reported by different algorithms for various datasets with differing sizes of pattern sets k .

Data set	Pattern	Search Algorithm									
	Set Size	Random Walk		Random Valid		Hill Climbing		HC + Restart		Genetic Algorithm	
	k	best	avg	best	avg	best	avg	best	avg	best	avg
zoo-1	2	100	66.87	82.13	73.93	94.15	82.74	100	92.07	100	98.06
	3	93.7	73.73	98.09	77.07	98.18	85.25	100	94.71	100	97
	4	96.92	76.97	92.17	78.19	91.97	84.4	94.76	89.71	100	96.71
hepatitis	2	89.17	71.7	86.06	76.43	87.19	78.18	100	92.13	100	97.49
	3	82.07	78.91	83.04	77.52	95.76	82.96	100	91.93	100	94.7
	4	89.47	75.24	87.48	77.57	100	86.52	100	93.13	100	96.24
lymph	2	81.4	76.51	100	78.35	97.4	81.94	100	92.97	100	96.72
	3	83.41	71.17	98.47	75.64	91.67	82.07	100	90.85	100	95.04
	4	85.67	72.62	86.89	82.29	89.54	80.56	92.38	90.42	100	95.7
heart-cleveland	2	96.69	77.35	84.27	75.71	100	86.3	100	89.34	100	96.89
	3	100	82.75	90.78	70.31	97.51	85.28	97.87	85.8	100	95.21
	4	98.78	76.71	96.4	78.1	97.56	83.41	100	91.38	100	93.29
vote	2	83.41	71.41	85.21	71.44	98.42	81.78	100	89.76	100	94.84
	3	85.54	72.3	87.4	73.79	87.41	80.34	96.12	89.6	100	93.84
	4	78.05	68.82	87.08	77.97	84.91	80.48	98.84	89.84	98.46	94.48
primary-tumor	2	100	80.75	88.69	72.61	85.54	78.37	96.47	87.52	100	93.25
	3	88.69	72.11	78.46	72.65	87.59	79.69	97.87	86.81	100	91.41
	4	82.26	66.64	84.65	70.91	89.54	82.11	98.57	87.48	100	93.14

of best accuracy achieved.

6 Conclusion

In this paper, we showed the effectiveness of various stochastic local search algorithms to solve the concept learning task in pattern set mining. Population based algorithms like genetic algorithm shows promising results while local minima escaping strategies like random restart increases the effectiveness of greedy hill climbing search to a great extent. In future, we would like to improve the performance of the search techniques by incorporating further effective strategies within the framework of stochastic local search and solve pattern set mining related problems with realistic datasets.

References

- [1] B. Bringmann, S. Nijssen, N. Tatti, J. Vreeken, and A. Zimmerman, “Mining sets of patterns,” *Tutorial at ECMLP-KDD*, 2010.

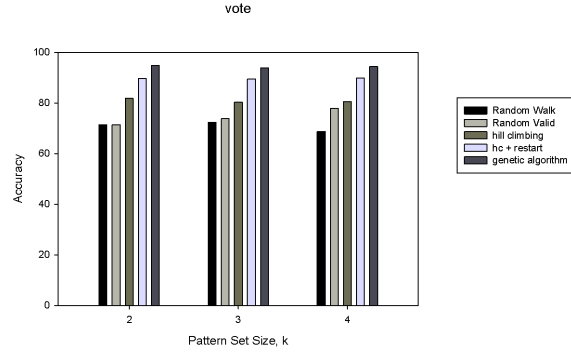


Figure 5: Put a title here

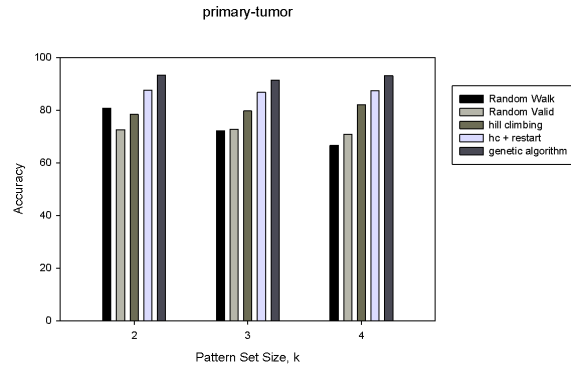


Figure 6: Put a title here

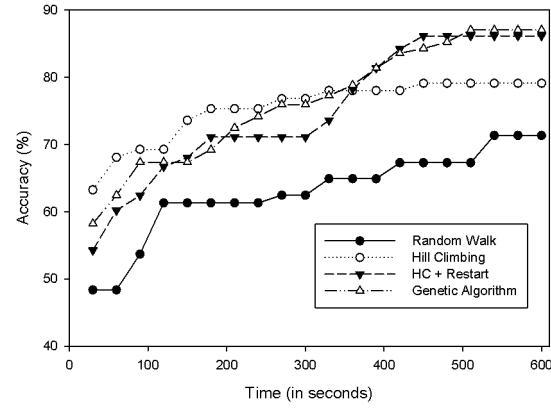


Figure 7: Search progress for different algorithm for the hepatitis dataset with size of the pattern size $k = 3$.

- [2] T. Guns, S. Nijssen, A. Zimmermann, and L. De Raedt, “Declarative heuristic search for pattern set mining,” in *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*. IEEE, 2011, pp. 1104–1111.
- [3] T. Guns, S. Nijssen, and L. De Raedt, “Itemset mining: A constraint programming perspective,” *Artificial Intelligence*, vol. 175, no. 12, pp. 1951–1983, 2011.
- [4] T. Guns, S. Nijssen, and L. D. Raedt, “k-pattern set mining under constraints,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 25, no. 2, pp. 402–418, 2013.
- [5] R. Agrawal, T. Imieliński, and A. Swami, “Mining association rules between sets of items in large databases,” in *ACM SIGMOD Record*, vol. 22, no. 2. ACM, 1993, pp. 207–216.
- [6] D. J. Hand, *Pattern detection and discovery*. Springer, 2002.
- [7] S. Morishita and J. Sese, “Transversing itemset lattices with statistical metric pruning,” in *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2000, pp. 226–236.
- [8] H. Cheng, X. Yan, J. Han, and C.-W. Hsu, “Discriminative frequent pattern analysis for effective classification,” in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 2007, pp. 716–725.
- [9] P. K. Novak, N. Lavrač, and G. I. Webb, “Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining,” *The Journal of Machine Learning Research*, vol. 10, pp. 377–403, 2009.
- [10] U. Rückert and S. Kramer, “Optimizing feature sets for structured data,” in *Machine Learning: ECML 2007*. Springer, 2007, pp. 716–723.
- [11] P. V. Hentenryck and L. Michel, *Constraint-based local search*. The MIT Press, 2009.
- [12] L. De Raedt, “Declarative modeling for machine learning and data mining,” in *Formal Concept Analysis*. Springer, 2012, pp. 2–2.
- [13] T. Guns, A. Dries, G. Tack, S. Nijssen, and L. De Raedt, “Miningzinc: A modeling language for constraint-based mining,” in *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*. AAAI Press, 2013, pp. 1365–1372.
- [14] A. Frank, A. Asuncion *et al.*, “UCI machine learning repository,” 2010.